MAVEN DEPENDENCY MANAGEMENT

One of the core features of Maven is Dependency Management. Managing dependencies become difficult task once we've to deal with multi-module projects (consists of hundreds of modules/sub-projects). Maven provides a high degree of control to manage such scenarios.

Transitive Dependencies Discovery

It is pretty often a case, when a library say A depends upon other library say B. In case another project C want to use A then that project requires to use library B too.

Maven helps to avoid such requirement to discover all the libraries required. Maven does so by reading project files(pom.xml) of dependencies, figure out their dependencies and so on.

We only need to define direct dependency in each project pom. Maven handles the rest automatically.

With transitive dependencies, the graph of included libraries can quickly grow to a large extent.Cases can arise when there are duplicate libraries. Maven provides few features to control extent of transitive dependencies

Feature	Description
Dependency mediation	Determines what version of a dependency is to be used when multiple versions of an artifact are encountered. If two dependency versions are at the same depth in the dependency tree, the first declared dependency will be used.
Dependency management	Directly specify the versions of artifacts to be used when they are encountered in transitive dependencies. For an example project C can include B as a dependency in its dependencyManagement section and directly control which version of B is to be used when it is ever referenced.
Dependency scope	Includes dependencies as per the current stage of the build
Excluded dependencies	Any transitive dependency can be excluede using "exclusion" element. As example, A depends upon B and B depends upon C then A can mark C as excluded.
Optional dependencies	Any transitive dependency can be marked as optional using "optional" element. As example, A depends upon B and B depends upon C. Now B marked C as optional. Then A will not use C.

Dependency Scope

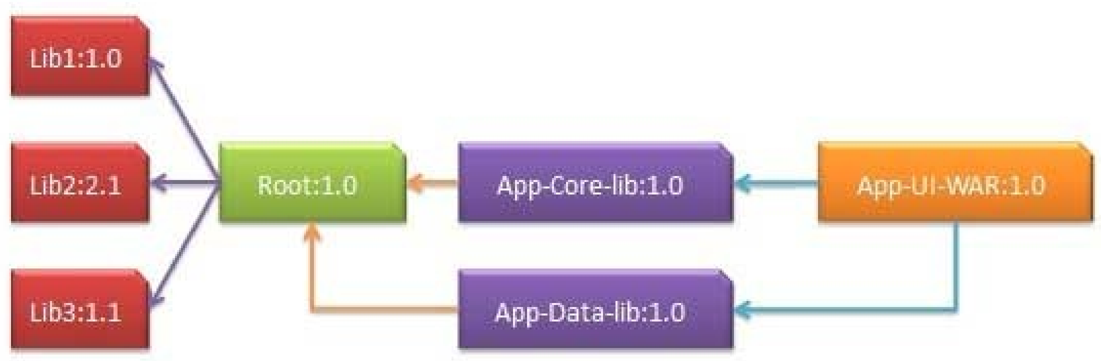
Transitive Dependencies Discovery can be restricted using various Dependency Scope as mentioned below

Scope	Description
compile	This scope indicates that dependency is available in classpath of project. It is default scope.
provided	This scope indicates that dependency is to be provided by JDK or web-Server/Container at runtime.

runtime	This scope indicates that dependency is not required for compilation, but is required during execution.
test	This scope indicates that the dependency is only available for the test compilation and execution phases.
system	This scope indicates that you have to provide the system path.
import	This scope is only used when dependency is of type pom. This scopes indicates that the specified POM should be replaced with the dependencies in that POM's <dependencyManagement> section.

Dependency Management

Usually, we've a set of project under a common project. In such case, we can create a common pom having all the common dependencies and then make this pom parent of sub-project's poms. Following example will help you understand this concept



Following are the details of above dependency graph

- App-UI-WAR depends upon App-Core-lib and App-Data-lib.
- Root is parent of App-Core-lib and App-Data-lib.
- Root defines Lib1,lib2, Lib3 as dependencies in its dependency section.

App-UI-WAR

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
   http://maven.apache.org/xsd/maven-4.0.0.xsd">
      <modelVersion>4.0.0</modelVersion>
      <groupId>com.companyname.groupname</groupId>
      <artifactId>App-UI-WAR</artifactId>
      <version>1.0</version>
      <packaging>war</packaging>
      <dependencies>
         <dependency>
            <groupId>com.companyname.groupname</groupId>
            <artifactId>App-Core-lib</artifactId>
            <version>1.0</version>
         </dependency>
      </dependencies>
      <dependencies>
         <dependency>
            <groupId>com.companyname.groupname</groupId>
            <artifactId>App-Data-lib</artifactId>
            <version>1.0</version>
         </dependency>
      </dependencies>
</project>
```

App-Core-lib

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
   http://maven.apache.org/xsd/maven-4.0.0.xsd">
      <parent>
         <artifactId>Root</artifactId>
         <groupId>com.companyname.groupname</groupId>
         <version>1.0</version>
      </parent>
      <modelVersion>4.0.0</modelVersion>
      <groupId>com.companyname.groupname</groupId>
      <artifactId>App-Core-lib</artifactId>
      <version>1.0</version>
      <packaging>jar</packaging>
</project>
```

App-Data-lib

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
   http://maven.apache.org/xsd/maven-4.0.0.xsd">
      <parent>
         <artifactId>Root</artifactId>
         <groupId>com.companyname.groupname</groupId>
         <version>1.0</version>
      </parent>
      <modelVersion>4.0.0</modelVersion>
      <groupId>com.companyname.groupname</groupId>
      <artifactId>App-Data-lib</artifactId>
      <version>1.0</version>
      <packaging>jar</packaging>
</project>
```

Root

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
   http://maven.apache.org/xsd/maven-4.0.0.xsd">
      <modelVersion>4.0.0</modelVersion>
      <groupId>com.companyname.groupname</groupId>
      <artifactId>Root</artifactId>
      <version>1.0</version>
   <packaging>pom</packaging>
      <dependencies>
         <dependency>
            <groupId>com.companyname.groupname1</groupId>
            <artifactId>Lib1</artifactId>
            <version>1.0</version>
         </dependency>
      </dependencies>
      <dependencies>
         <dependency>
            <groupId>com.companyname.groupname2</groupId>
            <artifactId>Lib2</artifactId>
            <version>2.1</version>
         </dependency>
      </dependencies>
      <dependencies>
         <dependency>
            <groupId>com.companyname.groupname3</groupId>
            <artifactId>Lib3</artifactId>
            <version>1.1</version>
         </dependency>
      </dependencies>
</project>
```

Now when we build App-UI-WAR project, Maven will discover all the dependencies by traversing the dependency graph and build the application.

From above example, we can learn following key concepts

- Common dependencies can be placed at single place using concept of parent pom.Dependencies of *App-Data-lib* and *App-Core-lib* project are listed in *Root* project (See the packaging type of Root. It is POM).
- There is no need to specify Lib1, lib2, Lib3 as dependency in App-UI-WAR. Maven use the **Transitive Dependency Mechanism** to manage such details.